Remarks

Claims 1, 29, 57, 63, 68 and 73 have been amended to recite "wherein the runtime security policy comprises an assignment of rights to the input component code and security checks performed as the input component code is loaded". Support for this amendment can be found, for example, at page 6, lines 16-20 of the Specification. No new matter has been introduced by this amendment.

I. Rejection of Claims 73-77 Under 35 U.S.C. §101 As Being Directed to Non-Statutory Subject Matter

Claims 73-77 were rejected under 35 U.S.C. §101 as being directed to non-statutory subject matter. Claim 73 has been amended in accordance with the Examiner's suggestion. Accordingly, withdrawal of this rejection is respectfully requested.

II. Rejection of Claims 1-4, 27-32 and 55-58 Under 35 U.S.C. §102(e) As Being Anticipated by U.S. Patent Number 7,051,322 to Rioux

Claims 1-4, 27-32 and 55-58 were rejected under 35 U.S.C. §102(e) as being anticipated by U.S. Patent Number 7,051,322 to Rioux ("Rioux"). Applicant respectfully traverses the rejection as follows.

Independent Claim 1

Claim 1 is directed to a method. Claim 1 recites receiving into an execution environment input component code and a runtime security policy; and generating a call graph of call paths through the input component code simulated in combination with at least one symbolic component representing additional arbitrary code that complies with the runtime security policy. Rioux does not teach, suggest or make obvious these features.

Rioux teaches a decompilation method of operation and system for parsing executable code, identifying and recursively modeling data flows, identifying and recursively modeling control flow, and iteratively refining these models to provide a complete model at the nanocode level. (Rioux, Abstract).

Regarding claim 1, in response to Applicant's previous argument that Rioux fails to disclose "receiving a runtime security policy", the Office Action provides:

> Applicant repeats arguments to this feature and Examiner maintains that Applicant's own Specification recites that "various security checks performed as the code is loaded … referred to as the runtime security policy."
>
> See Applicant's Specification, page 6, lines 11-14, "the **assignment of rights to code** and **various security checks** performed as the code is loaded are referred to as the runtime security policy." Rioux (col. 3:66-67), "the load and unlinker **read ("load") the target executable code into memory** (receiving a runtime security policy) and unlink the various segments of code…" Rioux (col. 11:3-11)… (Office Action at pages 3 and 4, emphasis in original).

As amended herein, claim 1 recites receiving into an execution environment input component code and a runtime security policy, wherein the runtime security policy comprises an assignment of rights to the input component code and security checks performed as the input component code is loaded. The portion of Rioux relied upon in the Office Action as teaching a runtime security policy provides:

> The loaded and unlinker read ("load") the target executable code into memory and unlink the various segments of code from one another through standard methods known in the art to produce stand-alone modules according to the organization of the executable code. (Rioux, col. 3, line 66 – col. 4, line 4).

Rioux further teaches:

> Intermediate representations of modeled executable code can thus be scanned or analyzed for flaws or conditions, especially including security holes, buffer structure flaws exploitable via "buffer overflow" attack, and other known and unknown risk factors. Such use is of great interest in the software arts today as a means of certifying software as trusted and/or determining whether software is safe to operate in mission-critical applications, for example. (Rioux, col. 11, lines 3-10).

Thus, Rioux teaches loading of the target executable and unlinking of the various segments to produce stand-alone modules. Rioux further teaches analysis for flaws or conditions such as security holes and buffer structure flows exploitable via "buffer overflow" attack. However, Rioux does not teach, disclose or suggest receiving a runtime security policy.

Rioux does not teach, suggest or make obvious receiving into an execution environment input component code and a runtime security policy, wherein the runtime security policy comprises an assignment of rights to the input component code and security checks performed as the input component code is loaded as recited in claim 1.

Since claim 1 recites features not taught or suggested by the reference of record, claim 1 patentably distinguishes over the reference of record and is in condition for allowance. Furthermore, dependent claims 2-28 also patentably distinguish over the reference of record and are in condition for allowance.

Independent claim 29

Claim 29 is directed to a computer program storage medium encoding a computer program for executing on a computer system a computer process. Claim 29 recites receiving into an execution environment input component code and a runtime security policy, wherein the runtime security policy comprises an assignment of rights to the input component code and security checks performed as the input component code is loaded; and generating a call graph of call paths through the input component code simulated in combination with at least one symbolic component representing additional arbitrary code that complies with the runtime security policy. Rioux does not teach, suggest or make obvious these features.

As discussed above, Rioux teaches a decompilation method of operation and system for parsing executable code, identifying and recursively modeling data flows, identifying and recursively modeling control flow, and iteratively refining these models to provide a complete model at the nanocode level. (Rioux, Abstract).

The Office Action provides the same response to Applicant's previous argument that Rioux fails to disclose "receiving a runtime security policy" and the same basis for rejecting claim 29 as discussed regarding claim 1 above.

As amended herein, claim 29 recites receiving into an execution environment input component code and a runtime security policy, wherein the runtime security policy comprises an assignment of rights to the input component code and security checks performed as the input component code is loaded. As explained in greater detail with respect to claim 1, Rioux teaches loading of the target executable and unlinking of the various segments to produce stand-alone modules. Rioux further teaches analysis for flaws or conditions such as security holes and buffer structure flows exploitable via "buffer overflow" attack. However, Rioux does not teach, disclose or suggest receiving a runtime security policy.

Rioux does not teach, suggest or make obvious receiving into an execution environment input component code and a runtime security policy, wherein the runtime security policy comprises an assignment of rights to the input component code and security checks performed as the input component code is loaded as recited in claim 29.

Since claim 29 recites features not taught or suggested by the reference of record, claim 29 patentably distinguishes over the reference of record and is in condition for allowance. Furthermore, dependent claims 30-56 also patentably distinguish over the reference of record and are in condition for allowance.

Independent claim 57

Claim 57 is directed to a system. Claim 57 recites a processing unit and a call graph generator executing on the processing unit that receives into an execution environment input component code and a runtime security policy and generates a call graph of call paths through the input component code simulated in combination with at least one symbolic component that represents additional arbitrary code that complies with the runtime security policy, wherein the runtime security policy comprises an assignment of rights to the input component code and security checks performed as the input component code is loaded. Rioux does not teach, suggest or make obvious these features.

As discussed above, Rioux teaches a decompilation method of operation and system for parsing executable code, identifying and recursively modeling data flows, identifying and recursively modeling control flow, and iteratively refining these models to provide a complete model at the nanocode level. (Rioux, Abstract).

The Office Action provides the same response to Applicant's previous argument that Rioux fails to disclose "receiving a runtime security policy" and the same basis for rejecting claim 57 as discussed regarding claim 1 above.

As amended herein, claim 57 recites a call graph generator executing on the processing unit that receives into an execution environment input component code and a runtime security policy and generates a call graph of call paths through the input component code simulated in combination with at least one symbolic component that represents additional arbitrary code that complies with the runtime security policy, wherein the runtime security policy comprises an assignment of rights to the input component code and security checks performed as the input component code is loaded.

As explained in greater detail with respect to claim 1, Rioux teaches loading of the target executable and unlinking the various segments to produce stand-alone modules. Rioux further teaches analysis for flaws or conditions such as security holes and buffer structure flows exploitable via "buffer overflow" attack. However, Rioux does not teach, disclose or suggest receiving a runtime security policy.

Rioux does not teach, suggest or make obvious a call graph generator executing on the processing unit that receives into an execution environment input component code and a runtime security policy and generates a call graph of call paths through the input component code simulated in combination with at least one symbolic component that represents additional arbitrary code that complies with the runtime security policy, wherein the runtime security policy comprises an assignment of rights to the input component code and security checks performed as the input component code is loaded as recited in claim 57.

Since claim 57 recites features not taught or suggested by the reference of record, claim 57 patentably distinguishes over the reference of record and is in condition for allowance. Furthermore, dependent claims 58-62 also patentably distinguish over the reference of record and are in condition for allowance.

III. Rejection of Claims 5-26, 33-54 and 59-77 Under 35 U.S.C. §103(a) As Being Obvious Over Rioux in view of U.S. Published Application Number 2005/0010806 to Berg

Claims 5-26, 33-54 and 59-77 were rejected under 35 U.S.C. §103(a) as being obvious over Rioux in view of U.S. Published Application Number 2005/0010806 to Berg ("Berg"). Applicant respectfully traverses the rejection of these claims.

Dependent claims 5-26, 33-54 and 59-62

Claims 5-26, 33-54 and 59-62 are dependent claims and are allowable based on their dependency from allowable independent claims as described above. Accordingly, Applicant respectfully requests that the rejection of these claims be withdrawn.

Independent claims 63, 68 and 73

Claim 63 is directed to a method, claim 68 is directed to a computer storage medium, and, claim 73 is directed to a system. These independent claims recite analyzing relative to at least one query a call graph of call paths through input component code simulated in combination with at least one symbolic component representing additional arbitrary code that complies with a runtime security policy, wherein the runtime security policy comprises an assignment of rights to the input component code and security checks performed as the input component code is loaded; and identifying a subset of the call paths in the call graph that satisfy the query. Rioux and Berg, individually and/or in combination, fail to teach these features.

As discussed with respect to claim 1, Rioux does not teach, suggest or make obvious a runtime security policy wherein the runtime security policy comprises an assignment of rights to the input component code and security checks performed as the input component code is loaded. Berg does not cure the shortcomings of Rioux.

The Office Action provides:

> However, Berg disclosed [0276-0277] branches/ arbitrary control flow (subset of call paths/additional arbitrary code). Also, see rejections addressed in claims 1 and 19 above. (Office Action at page 28).

Applicant respectfully submits that Berg does not teach, suggest or make obvious a runtime security policy. Thus, Berg does not teach, suggest or make obvious Rioux does not teach, suggest or make obvious a runtime security policy wherein the runtime security policy comprises an assignment of rights to the input component code and security checks performed as the input component code is loaded as recited in claims 63, 68 and 73.

Since claims 63, 68 and 73 recite features not taught or suggested by the references of record, claims 63, 68 and 73 patentably distinguishes over the references of record and is in condition for allowance. Furthermore, dependent claims 64-67, 69-72 and 74-77 also patentably distinguish over the references of record and are in condition for allowance.

Conclusion

For the reasons set forth above, claims 1-77 patentably and unobviously distinguish over the references and are allowable. An early allowance of all claims is earnestly solicited.

Respectfully submitted,

Date: May 1, 2008

/Jeffrey R. Sadlowski/
Jeffrey R. Sadlowski , Reg. No. 47,914
(216) 925-5482
Jeffrey R. Sadlowski, LLC
Millside Centre, Suite 11
8803 Brecksville Road
Brecksville, Ohio 44141